A Performance Optimisation Strategy for Real-Time Data Pipelines Based on Kafka and Flink

Xiaoxi Liu Shuhao Dong

College of Information Engineering, Hainan Vocational University of Science and Technology,Haikou,HaiNan,570100;

Abstract: With the surging demand for real-time big data processing, Kafka-Flink-based real-time data pipelines have become core components of enterprise data architectures. However, in production environments, pipeline performance frequently encounters bottlenecks due to data skew, suboptimal resource allocation, and inadequately managed backpressure mechanisms. This paper aims to systematically analyse the key factors affecting the performance of Kafka-Flink real-time data pipelines and proposes a comprehensive end-to-end optimisation strategy covering data production, transmission, computation, and coordination. By optimising Kafka's topic partitioning and producer configuration, adjusting Flink's task scheduling and state management, and introducing dynamic backpressure awareness and resource elastic scaling mechanisms, the proposed strategy significantly enhances pipeline throughput, reduces processing latency, and improves stability. Finally, the effectiveness of the proposed strategy is validated through simulation experiments.

Keywords : real-time data pipeline; Apache Kafka; Apache Flink; performance optimisation; backpressure mechanism

DOI： 10.69979/3041-0843.25.04.082

1 Introduction

1.1 Research Background and Significance

Presently, enterprise business decision-making demands increasingly stringent data timeliness requirements. The shift from traditional T+1 offline analysis models towards real-time insights at the second or even millisecond level has become an inevitable trend. Against this backdrop, real-time data pipeline technologies capable of continuously processing massive volumes of streaming data have become critically important. Apache Kafka, as a high-throughput distributed message queue, and Apache Flink, as a high-performance distributed stream processing engine, form an industry-recognised golden combination for real-time data processing when integrated. However, constructing a high-performance, highly available Kafka-Flink data pipeline is no simple task. Its performance is constrained by multiple factors, including data sources, networking, serialisation, resource management, and computational logic complexity. Therefore, systematically researching its performance optimisation strategies holds significant theoretical and practical value for ensuring the stability of real-time operations, reducing infrastructure costs, and fully unlocking the real-time value of data.

1.2 Research Status and Existing Challenges

Scholars and industry experts worldwide have extensively explored optimising the individual performance of Kafka and Flink. Regarding Kafka, research has predominantly focused on optimising partitioning strategies, replica synchronisation mechanisms, producer batching, and compression algorithms. For Flink, research emphasis has been placed on state backend selection, checkpoint mechanisms, window operator optimisation, and memory management. However, existing studies predominantly treat Kafka and Flink as two independent systems for optimisation, lacking an end-to-end data pipeline perspective and overlooking their strong coupling and mutual influence. For instance, mismatches between Kafka's production rate and Flink's consumption capacity can trigger backpressure. Simple throttling or scaling strategies may fail to address the root cause and could even lead to resource wastage. This paper therefore aims to address this shortcoming by proposing a global performance optimisation framework that synergises Kafka and Flink.

2 Performance Optimisation Strategies at the Kafka Layer

2.1 Producer and Topic Partition Optimisation

The performance of Kafka producers directly determines the rate of the data ingestion pipeline. Optimisation hinges on reducing the number of network requests and increasing the payload per request. Specific strategies include: enabling producer-side batching mechanisms by judiciously configuring parameters such as `linger.ms` (e.g., 10 – 100ms) and `batch.size` (e.g., 16 – 64KB) to balance latency and throughput; employing efficient serialisers like Apache Avro alongside Schema Registry to reduce data volume; and activating compression algorithms such as Snappy or LZ4 to alleviate network and storage I/O pressure. At the topic partition level, the number of partitions underpins Kafka's parallelism. Ensure partition count does not fall below the parallelism of Flink consumer tasks to prevent idle Flink tasks. Partitioning strategies should prevent data skew; for keyed data, employ consistent hashing or similar techniques to ensure even data distribution across partitions, laying the foundation for balanced load in downstream Flink parallel computations.

2.2 Cluster Configuration and Consumer-Side Tuning

The configuration of the Kafka cluster itself has a decisive impact on performance. Firstly, the acknowledgement mechanism must be set based on a trade-off between data reliability and performance. For scenarios where latency is extremely sensitive and a small amount of data loss is tolerable, it can be set to 0 or 1. For scenarios requiring high reliability, it should be set to all and min.insync.replicas should be configured appropriately. Secondly, optimise log cleaning strategies by selecting time-based or size-based policies according to data retention requirements, while monitoring disk usage to prevent fill-up. On the consumer side (i.e., Flink's Kafka Consumer), optimisation focuses on enhancing consumption efficiency and fault tolerance. It is advisable to set appropriate fetch.min.bytes and fetch.max.wait.ms values to improve fetch efficiency; disable automatic offset commit, instead having Flink commit uniformly during checkpoints to achieve precise once semantics; and dynamically adjust the consumption rate based on the actual processing capacity of the Flink task to avoid buffer overflows in Flink operators due to excessive consumption or Kafka backlogs caused by insufficient consumption.

3 Flink Layer Performance Optimisation Strategies

3.1 Task Parallelism and Resource Allocation

The core of Flink job performance lies in the reasonable configuration of parallelism and effective resource allocation. The job parallelism should maintain a multiple relationship with the number of Kafka topic partitions to ensure all Flink task instances receive data. For different operators, specific parallelism can be set based on their computational density (e.g., CPU-intensive or I/O-intensive) to prevent any single link in the chain of operators from becoming a performance bottleneck. Regarding resource allocation, appropriate memory must be assigned to TaskManager containers within resource managers like YARN or Kubernetes. Key adjustments involve balancing the proportions of task heap memory, managed memory (for RocksDB state backends), network buffers, and JVM metaspace within Flink's total memory. Insufficient managed memory is a common cause of RocksDB performance degradation; ensure sufficient memory is reserved based on state size. Simultaneously, ensure each TaskManager has a reasonable number of slots, avoiding excessive slots that cause resource fragmentation or insufficient slots that lead to resource wastage.

3.2 State Management and Checkpoint Mechanism

State management forms the core of Flink's stream processing, with its performance directly impacting pipeline stability and latency. For scenarios involving large states, the RocksDB state backend is strongly recommended. It stores state on disk, limited only by local disk capacity, making it suitable for massive state volumes. Additionally, enabling state TTL functionality allows automatic cleanup of expired state, preventing infinite state growth. The checkpoint mechanism underpins Flink's fault tolerance, though its creation incurs performance overhead. Optimisation measures include: Employing incremental checkpoints over full checkpoints, particularly advantageous with RocksDB, significantly reducing the data volume persisted per checkpoint; Adjusting checkpoint intervals to balance fault recovery speed against runtime

overhead (e.g., 1-5 minutes); For scenarios demanding ultra-low latency with at-least-once semantics tolerance, consider enabling Unaligned Checkpoint to mitigate backpressure's impact on checkpoint completion time, though this may increase state storage volume.

4 End-to-End Coordination and Advanced Optimisation Strategies

4.1 Dynamic Backpressure Awareness and Adaptive Control

Backpressure is an inherent phenomenon in stream processing systems, indicating downstream processing cannot keep pace with upstream production. Conventional backpressure handling may cause checkpoint timeouts or even job failures. This paper proposes a dynamic backpressure awareness and adaptive control strategy. Firstly, by monitoring Flink job backpressure metrics (Netty output queue length, backpressure ratio, etc.), operators generating backpressure are identified in real time. Secondly, this information is fed back to the pipeline's source, namely the Kafka producer. In practical implementation, a custom Metric Reporter can be developed on the Flink side to convert backpressure intensity into a control signal. This signal is then transmitted via a lightweight message channel (such as another Kafka Topic) to an adaptive controller. This controller dynamically adjusts the Kafka producer's transmission rate (e.g., by modifying linger.ms or batch.size) or samples and dilutes non-critical data. This system-level smoothing of data flow prevents cascading backpressure deterioration, enabling adaptive flow control.

4.2 Resource Elastic Scaling and Data Skew Handling

Static resource configuration struggles to accommodate fluctuations in streaming data volumes, making resource elastic scaling crucial for balancing cost and efficiency. An automated scaling strategy can be constructed based on the partition message lag metric for Kafka topics. When lag persistently exceeds a threshold, this automatically triggers an increase in Flink job parallelism or expansion of TaskManager instances; When Lag falls below the threshold, automatic scaling down conserves resources. This process may integrate with cloud platform elastic scaling services. For data skew issues, beyond optimising Kafka partitioning, Flink employs local aggregation (Combiner) to mitigate skew after KeyBy operations; for windowed aggregation, a two-stage approach may be attempted (first breaking down local aggregates, then performing global summarisation). For severe, unavoidable key skew, consider using the rebalance() operator to randomly redistribute data when business logic permits. This breaks key constraints, sacrificing locality for overall throughput improvement.

5 Experimental Validation and Conclusions

5.1 Experimental Design and Results Analysis

To validate the effectiveness of the aforementioned optimisation strategies, we established an experimental environment comprising a three-node Kafka cluster and a Flink on YARN cluster consisting of four servers. A custom data generator simulated JSON-formatted user behaviour logs. A typical Flink pipeline was constructed, encompassing a Kafka data source, data cleansing, KeyBy, window aggregation, and writing to a downstream database. The experiment comprised a baseline group (using default configurations) and an optimised group (applying the full suite of strategies proposed herein). Through load testing, we compared the two groups' throughput (records per second), end-to-end latency (P99), and system resource utilisation (CPU, memory) within a fixed time window. Results demonstrate that the optimised group achieved approximately 2.8 times higher throughput than the baseline group, with P99 latency reduced by 65%. Furthermore, during simulated data surges, the adaptive control mechanism rapidly stabilised the system without checkpoint timeouts, validating the comprehensiveness and effectiveness of the optimisation strategies.

5.2 Conclusions and Future Directions

This paper systematically analyses performance bottlenecks within Kafka-Flink real-time data pipelines, proposing a comprehensive end-to-end optimisation strategy spanning Kafka production transmission, Flink computation, and collaborative processing. This approach emphasises treating Kafka and Flink as an integrated whole for tuning, particularly through dynamic backpressure awareness and adaptive control mechanisms, offering novel solutions to the core challenge of

backpressure management. Experiments demonstrate that this strategy significantly enhances pipeline throughput, reduces processing latency, and improves system stability. Future work will focus on intelligent parameter tuning, exploring reinforcement learning-based models for automatic parameter optimisation. Concurrently, research will investigate how to achieve finer-grained elastic scaling and cost control of computational resources within hybrid cloud or serverless architectures, further reducing operational complexity and total cost of ownership for real-time data pipelines.

References

[1] Chen Xi, Li Jinsong. Research on Performance Optimisation of Kafka-Based Real-Time Data Streaming Platforms [J]. Computer Engineering, 2021, 47(5): 26-33.
[2] Wang Lei, Zhao Zhigang. Practical Optimisation of Apache Flink State Management and Checkpoint Mechanisms [J]. Journal of Software, 2020, 31(Supplement 1): 112-120.
[3] Zhang Jun, Liu Yang. Handling Strategies for Data Skew Issues in Large-Scale Stream Processing [J]. Research and Development in Computer Science, 2019, 56(11): 2450-2460.
[4] Guo Qiang, Gao Hong. Performance Tuning of Kafka Clusters for Real-time Data Pipelines [J]. Journal of East China Normal University (Natural Science Edition), 2022, (3): 45-55.
[5] Kreps, J. (Author). Translated by Dan Wei et al. Kafka: The Definitive Guide [M]. Beijing: People's Posts and Telecommunications Press, 2017.